Electronic Notes in Theoretical Computer Science 35 (2000)
URL: http://www.elsevier.nl/locate/entcs/volume35.html 18 pages

Soundness of a purely syntactical formalization of weakest preconditions

Rudolf Berghammer

Institut für Informatik und Praktische Mathematik
Christian-Albrechts-Universität Kiel
Preusserstraße 1-9, 24105 Kiel, Germany
e-mail: rub@informatik.uni-kiel.de

Abstract

We present a purely syntactical definition of E.W. Dijkstra's predicate transformer *wp* for nondeterministic while-programs in infinitary logic. Then we show that it is sound with respect to a definition of weakest preconditions given in terms of denotational semantics.

1 Introduction

Usually, semantics of programming languages comes in three flavours, viz. operational, denotational, and axiomatic. Of the latter, one of the best known is weakest precondition semantics (also known as *wp*-calculus), introduced by E.W. Dijkstra [9,10] as a generalization of Hoare logic [16] to reasoning about total correctness of imperative programs. It is based on a function *wp* which, usually, is defined in a semantical fashion, i.e., maps a program p (the only syntactic argument) and a set of states R (the postcondition) to the largest set of states Q (the weakest precondition) such that execution of p from a state of Q will result in p terminating in a state of R.

Besides axiomating the semantics of a programming language, weakest preconditions can also be used as an aid in program construction. This has been shown by many authors including E.W. Dijkstra [10,11], D. Gries [14], R. Backhouse [5], and C. Morgan [19] for example. The techniques used here are based on predicate calculus and easily accessible to anyone with background in logic. Instead of working with sets of states or – as it is common practice – equating a logical formula with a predicate on states or the set of all states in which it is valid, therefore, a purely syntactical definition of *wp* as a function that maps a program and a formula to a formula should be used. This avoids inaccuracies and ambiguities caused by identifying formulae and their interpretations or by borrowing symbols from logic while using them for

something entire different. That the common practice may lead to wrong results is e.g., demonstrated in [24]. Our approach is also in accordance with the use of predicate logic as a general mathematical tool for writing formal specifications.

Purely syntactical definitions of the function *wp* are very rare. In this article we revisite R. Back's approach [2], where *wp* is – as a direct translation of E.W. Dijkstra's original formulation into logic – defined as a function on programs and formulae of the infinitary logic $\mathcal{L}_{\omega_1\omega}$. We present a rigorous syntactical definition of *wp* for nondeterministic while-programs. In doing so, the variable substitution problem of $\mathcal{L}_{\omega_1\omega}$ raised by assignment statements is solved by restricting the class of formulae. (In [2] this problem is neglected.) Our main result is that the presented syntactical definition of the function *wp* is sound wrt. a definition of weakest preconditions given in terms of denotational semantics (and the validity of formulae according to states). This soundness theorem leads to a perspicuous denotational standard model of the *wp*-calculus which is in sharp contrast with the intuitive model in [2] being an opportunistic hybrid of logic, set theory, and operational semantics. From a practical point of view it is also crucial, since, for example, a programmer might use weakest preconditions to construct programs and to reason about their properties, whereas a compiler writer might use denotational semantics to implement the programming language.

A formal statement and proof (as subsequently presented) could not be found in the hitherto existing literature, i.e., the soundness theorem seems to be a "folk theorem" in the sense of D. Harel [15]. Perhaps the reason for this is that those authors that use denotational semantics investigate only Hoare logic, while those with a preference to weakest preconditions define semantics in an operational manner.

2 Nondeterministic While-Programs

In this section we define syntax and denotational semantics of a simple imperative programming language. We assume the reader to be familiar with the notions of a signature Σ and a Σ-algebra and the foundations of denotational semantics like directed sets, (flat) complete partial orders (briefly: cpos), strictness, monotonicity, continuity, and the least fixed point operator μ. Details on signatures and algebras can be found in [26]; for the foundations of denotational semantics we refer to the textbooks [21,25].

The syntactical basis of the programs consists of a signature $\Sigma = (S, C, F)$, where S, C, and F are the sets of sorts, constant symbols (each is assigned a sort), and function symbols (each is assigned a functionality), respectively, and a countable infinite set X of variables (each is assigned a sort again). It is supposed that S contains at least the sort *bool* for the truth values, C contains at least the constant symbol *false* of sort *bool* for falsity, and F contains at least the function symbol *not* with functionality $bool \to bool$ for negation.

The set $\mathcal{E}_m$ of expressions of sort $m \in S$ over Σ and X is inductively defined as usual. Now we can fix the syntax of our programming language:

Definition 2.1 *The set $\mathcal{PRG}$ of non-deterministic while-programs (or statements) is inductively defined as follows:*

(i) *The empty statement* `skip`*, the undefined statement* `abort`*, and the assignment statements $(x := t)$, where $x \in X$ is of sort m and $t \in \mathcal{E}_m$, are elements of $\mathcal{PRG}$.*

(ii) *If $b \in \mathcal{E}_{bool}$ and $s_1, s_2 \in \mathcal{PRG}$, then the nondeterministic choice $(s_1 \,[\!]\, s_2)$, the conditional* `if` b `then` s_1 `else` s_2 `fi`*, the composed statement $(s_1; s_2)$, and the while-loop* `while` b `do` s_1 `od` *are also elements of $\mathcal{PRG}$.*

We use the customary rules of simplication to drop parentheses. Furthermore we denote the set of variables contained in a program $p \in \mathcal{PRG}$ by $var(p)$.

A fixed Σ-algebra $A = ((m^A)_{m\in S}, (c^A)_{c\in C}, (f^A)_{f\in F})$ is used to determine the meaning of the elements of Σ. Each sort m is interpreted as a non-empty set m^A. Furthermore, each constant symbol c is interpreted as an element c^A and each function symbol f is interpreted as a partial function f^A, respecting the sort of c respective the functionality of f, of course. We choose the standard interpretations $\{tt, ff\}$ for the sort $bool$, ff for the constant symbol $false$, and $not^A(tt) = ff$, $not^A(ff) = tt$ for the function symbol not.

Define $i[m] = m^A \cup \{\bot_m\}$ by including a new pseudo-value in the interpretation of the sort m. Then State is the set of functions $\sigma : X \to \bigcup_{m\in S} i[m]$ which meet the condition that the variable x is of sort m iff $\sigma(x) \in i[m]$, and $\mathsf{State}^\bot = \mathsf{State} \cup \{\bot\}$ is the corresponding flat cpo. We call an element from State a defined state and $\bot$ the undefined state. Given $\sigma \in \mathsf{State}$, a variable $x \in X$ of sort m, and an element $u \in i[m]$ we denote by $\sigma[x \leftarrow u]$ the defined state which is exactly like σ except for the argument x. Here it yields the value u. Finally we write $[\![t]\!](\sigma)$ for the value of the expression $t \in \mathcal{E}_m$ according to $\sigma \in \mathsf{State}^\bot$. We omit here a formal inductive definition of the function $[\![t]\!] : \mathsf{State}^\bot \to i[m]$ and mention only the strictness property $[\![t]\!](\bot) = \bot_m$.

By $\mathbf{P}[\mathsf{State}^\bot]$ we denote the set of all non-empty subsets of $\mathsf{State}^\bot$ which are finite or contain the undefined state $\bot$. We model nondeterministic while-programs by monotone functions from $\mathsf{State}^\bot$ to $\mathbf{P}[\mathsf{State}^\bot]$. For that reason we have to define an approximation order on $\mathbf{P}[\mathsf{State}^\bot]$ which then induces the pointwise order on the functions. The appropriate relation is the so-called *Egli-Milner order* $\leq_{EM}$ given by

$$M \leq_{EM} N \quad \text{iff} \quad (\bot \notin M \text{ and } M = N) \text{ or } (\bot \in M \text{ and } M \setminus \{\bot\} \subseteq N)\,.$$

Since $\mathsf{State}^\bot$ is flat, the result is a cpo, called *Plotkin power domain* [22]. The least element of $\mathbf{P}[\mathsf{State}^\bot]$ is the singleton set $\{\bot\}$. If $\mathcal{M} \subseteq \mathbf{P}[\mathsf{State}^\bot]$ is a directed set and there is $M \in \mathcal{M}$ which does not contain the undefined state $\bot$, then the least upper bound $\bigsqcup \mathcal{M}$ of $\mathcal{M}$ is M. Otherwise, i.e., each $M \in \mathcal{M}$ contains $\bot$, we have $\bigsqcup \mathcal{M} = \bigcup_{M\in\mathcal{M}} M$.

We are now in a position to present the denotational semantics of our programming language. If we imagine a nondeterministic computation operationally as a sequence of states, then this semantics describes that one arbitrarily chooses the next state, even it leads to nontermination although termination is possible by another choice. This kind of nondeterminism is known as erratic nondeterminism.

Definition 2.2 *The denotational semantics of a nondeterministic while-program $p \in \mathcal{PRG}$ is a function $[\![p]\!] : \mathsf{State}^{\perp} \to \mathbf{P}[\mathsf{State}^{\perp}]$ which is inductively defined as follows:*

(i) *Empty statement:*

$$[\![\texttt{skip}]\!](\sigma) = \{\sigma\}$$

(ii) *Undefined statement:*

$$[\![\texttt{abort}]\!](\sigma) = \{\perp\}$$

(iii) *Assignment statement:*

$$[\![x := t]\!](\sigma) = \begin{cases} \{\sigma[x \leftarrow [\![t]\!](\sigma)]\} & : \quad [\![t]\!](\sigma) \neq \perp_m \\ \{\perp\} & : \quad [\![t]\!](\sigma) = \perp_m \end{cases}$$

(iv) *Nondeterministic choice:*

$$[\![s_1 \parallel s_2]\!](\sigma) = [\![s_1]\!](\sigma) \cup [\![s_2]\!](\sigma)$$

(v) *Conditional:*

$$[\![\texttt{if } b \texttt{ then } s_1 \texttt{ else } s_2 \texttt{ fi}]\!](\sigma) = \begin{cases} [\![s_1]\!](\sigma) & : \quad [\![b]\!](\sigma) = tt \\ [\![s_2]\!](\sigma) & : \quad [\![b]\!](\sigma) = ff \\ \{\perp\} & : \quad [\![b]\!](\sigma) = \perp_{bool} \end{cases}$$

(vi) *Composed statement:*

$$[\![s_1; s_2]\!](\sigma) = \bigcup_{\rho \in [\![s_1]\!](\sigma)} [\![s_2]\!](\rho)$$

(vii) *While-loop:*

$$[\![\texttt{while } b \texttt{ do } s \texttt{ od}]\!](\sigma) = \mu_\tau(\sigma)\,,$$

where the functional τ on the cpo of the monotone functions from $\mathsf{State}^{\perp}$ to $\mathbf{P}[\mathsf{State}^{\perp}]$ is defined by

$$\tau[h](\sigma) = \begin{cases} \bigcup_{\rho \in [\![s]\!](\sigma)} h(\rho) & : \quad [\![b]\!](\sigma) = tt \\ \{\sigma\} & : \quad [\![b]\!](\sigma) = ff \\ \{\perp\} & : \quad [\![b]\!](\rho) = \perp_{bool}\,. \end{cases}$$

The functional τ used in the semantics of the while-loop is continuous. The fixed point theorem for continuous functions on cpos yields $\mu_\tau = \bigsqcup_{i\in\mathbb{N}} \tau^i[\Omega]$, where $\Omega : \mathsf{State}^\perp \to \mathbf{P}[\mathsf{State}^\perp]$ sends each state to $\{\perp\}$.

3 Assertions and Weakest Preconditions

When specifying pre- and postconditions of imperative programs by logical formulae, normally one uses additional function symbols not contained in the programming language's signature. Therefore we suppose in the following such an extension Σ_e of the signature Σ of the last section to be given and denote the set of expressions of sort $m \in S$ over the original set X of variables and the extended signature Σ_e by $\mathcal{T}_m$.

As already mentioned in the introduction we use R. Back's [2] approach to formalize the *wp*-calculus in infinitary first-order logic $\mathcal{L}_{\omega_1\omega}$. We will here only present the most basic notions of this logic, for details see [17,18] for example. Let $\mathcal{L}$ denote the set of ordinary first-order formulae built over Σ_e and X, where the Boolean expressions from $\mathcal{T}_{bool}$ are the atomic formulae and the connectives $\neg$, $\vee$, $\wedge$, $\rightarrow$, $\leftrightarrow$ and quantifiers $\forall$, $\exists$ are used. The set $\mathcal{L}_{\omega_1\omega}$ of formulae extends $\mathcal{L}$ by allowing also *infinite disjunctions* $\bigvee\Phi$ and *infinite conjunctions* $\bigwedge\Phi$, where $\Phi \subseteq \mathcal{L}_{\omega_1\omega}$ is countable infinite. When applying this set to formalize the *wp*-calculus we have also to deal with undefined values of expressions. Therefore we allow in $\mathcal{L}_{\omega_1\omega}$ further formulae of the special form $def[t]$, where t is an expression, and call them *definedness tests*. As in the case of programs, $var(\varphi)$ denotes the set of variables contained in $\varphi \in \mathcal{L}_{\omega_1\omega}$.

The basis of $\mathcal{L}_{\omega_1\omega}$'s semantics is an extension of the Σ-algebra A of Section 2 to a Σ_e-algebra A_e. As in the case of the expressions from $\mathcal{E}_m$ we write $[\![t]\!](\sigma)$ for the value of $t \in \mathcal{T}_m$ according to the state $\sigma \in \mathsf{State}^\perp$ and assume again $[\![t]\!](\perp) = \perp_m$. Then the usual notion of validity of an $\mathcal{L}$-formula φ according to a *defined* state σ, written as $\models \varphi[\sigma]$ (where for Boolean expressions $\models b[\sigma]$ iff $[\![b]\!](\sigma) = tt$), generalizes in a straightforward way to $\mathcal{L}_{\omega_1\omega}$ by

$$\begin{aligned}
\models \textstyle\bigvee\Phi\,[\sigma] \quad &\text{iff} \quad \text{there exists } \varphi \in \Phi \text{ such that } \models \varphi[\sigma]\,,\\
\models \textstyle\bigwedge\Phi\,[\sigma] \quad &\text{iff} \quad \models \varphi[\sigma] \text{ for all } \varphi \in \Phi\,,\\
\models def[t][\sigma] \quad &\text{iff} \quad [\![t]\!](\sigma) \neq \perp_m\,.
\end{aligned}$$

We write $\models \varphi$ iff $\models \varphi[\sigma]$ for all $\sigma \in \mathsf{State}$. Later we are also concerned with the validity of a $\mathcal{L}_{\omega_1\omega}$-formula according to a set of defined states. Therefore we generalize the relation $\models$ to a relation $\models\!\!\models$ on $\mathcal{L}_{\omega_1\omega}$ and 2^{State} by

$$\models\!\!\models \varphi[M] \quad \text{iff} \quad \models \varphi[\sigma] \text{ for all } \sigma \in M\,.$$

For a purely syntactical approach to *wp*, for an assignment statement $x := t$ substitution $\varphi[t/x]$ of the variable x by the expression t in the formula

φ is required. Substitution is also indispensable for practical working with formulae in program development or verification, e.g., if a derivation involves the renaming of variables. Many applications of syntactical replacement can e.g., be found in the refinement calculus [19,20] which is based upon weakest preconditions.

For $\mathcal{L}$-formulae substitution is inductively defined, where the most complicated case is quantification $Q \in \{\forall, \exists\}$ since here it may be necessary to rename a bound variable:

$$(Q\,y : \varphi)[t/x] = \begin{cases} Q\,y : \varphi & : x \text{ not free in } Q\,y : \varphi \\ Q\,y : \varphi[t/x] & : x \text{ free in } Q\,y : \varphi \text{ and } y \text{ not in } t \\ Q\,z : \varphi[z/y][t/x] & : \text{otherwise.} \end{cases}$$

Here z is a "fresh" variable which does not occur neither in the formula $Q\,y : \varphi$ nor in the expression t, usually the first fresh variable in the assumed enumeration $x_0, x_1, \ldots$ of the set X of variables such that substitution becomes unique. Substitution in quantified $\mathcal{L}_{\omega_1\omega}$-formulae in the just described simple way is impossible if $var(\varphi) = X$ holds as for $\exists\, x_0 : \bigvee\{x_0 = x_i : i > 0\}$. Hence we consider for a formalization of the *wp*-calculus a subset of $\mathcal{L}_{\omega_1\omega}$-formulae for which it can be applied.

Definition 3.1 *A formula $\varphi \in \mathcal{L}_{\omega_1\omega}$ is said to be an assertion if $var(\varphi)$ is finite. The set of all assertions is denoted by $\mathcal{ASS}$.*

When substitution in assertions is defined as in the case of ordinary-first order formulae, as a consequence we get the well-known substitution property

$$\models \varphi[t/x][\sigma] \quad \text{iff} \quad \models \varphi[\sigma[x \leftarrow [\![t]\!](\sigma)]]\,,$$

which will be used in the proof of soundness in the next section. After these preparations we are now in a position to formalize the decisive notions of totally correctness and weakest preconditions in terms of the denotational semantics and the two validity relations.

Definition 3.2 *a) A while-program $p \in \mathcal{PRG}$ is said to be totally correct according to a precondition $\psi \in \mathcal{ASS}$ and a postcondition $\varphi \in \mathcal{ASS}$, abbreviated as $[\psi]\,p\,[\varphi]$, if $\models \psi[\sigma]$ implies $\bot \notin [\![p]\!](\sigma)$ and $\models\!\!\models \varphi[[\![p]\!](\sigma)]$ for all $\sigma \in \mathsf{State}$.*

b) Given a while-program $p \in \mathcal{PRG}$ and a postcondition $\varphi \in \mathcal{ASS}$, a formula $\alpha \in \mathcal{ASS}$ is called a weakest precondition (for total correctness) if (i) $[\alpha]\,p\,[\varphi]$ and (ii) $[\psi]\,p\,[\varphi]$ implies $\models \psi \to \alpha$ for all $\psi \in \mathcal{ASS}$.

It should be mentioned that in Definition 3.2.a we suppose a sequential version of conjunction. I.e., $\bot \in [\![p]\!](\sigma)$ implies that the conjunction "$\bot \notin [\![p]\!](\bot)$ and $\models\!\!\models \varphi[[\![p]\!](\bot)]$" is false although $\models\!\!\models \varphi[[\![p]\!](\bot)]$ is not defined. A sequential

interpretation of conjunction can be avoided if additionally every formula is defined to be valid according to $\perp$ as, for example, G. Winskel [25] does.

4 Soundness of the wp-Calculus

E.W. Dijkstra [9,10] introduced the nondeterministic programming language of guarded commands and, for deriving programs, the *wp*-calculus as a generalization of Hoare logic [16] to total correctness. Its axioms and rules can be formalized in the infinitary logic $\mathcal{L}_{\omega_1\omega}$ as shown by R. Back [2]. The basis of the *wp*-calculus is an inductive definition of a function *wp* for computing weakest preconditions from programs and postconditions. Using assertions instead of general infinite first-order formulae, for the nondeterministic while-programs induced in Section 2 a purely syntactical version of E.W. Dijkstra's *wp*-function looks as follows.

Definition 4.1 *The function* $wp : \mathcal{PRG} \times \mathcal{ASS} \to \mathcal{ASS}$ *is inductively defined by:*

(i) Empty statement*:*

$$wp(\texttt{skip}, \varphi) = \varphi$$

(ii) Undefined statement*:*

$$wp(\texttt{abort}, \varphi) = \mathit{false}$$

(iii) Assignment*:*

$$wp(x := t, \varphi) = \mathit{def}\,[t] \wedge \varphi[t/x]$$

(iv) Nondeterministic choice*:*

$$wp(s_1 \,[\!]\, s_2, \varphi) = wp(s_1, \varphi) \wedge wp(s_2, \varphi)$$

(v) Conditional statement*:*

$$wp(\texttt{if } b \texttt{ then } s_1 \texttt{ else } s_2 \texttt{ fi}, \varphi) = (b \wedge wp(s_1, \varphi)) \vee (\mathit{not}(b) \wedge wp(s_2, \varphi))$$

(vi) Composed statement*:*

$$wp(s_1; s_2, \varphi) = wp(s_1, wp(s_2, \varphi))$$

(vii) While-loop*:*

$$wp(\texttt{while } b \texttt{ do } s \texttt{ od}, \varphi) = \bigvee \{\varphi_i : i \in \mathbb{N}\}\,,$$

where the countable infinite set $\{\varphi_i : i \in \mathbb{N}\}$ *of formulae is inductively defined by* $\varphi_0 = \mathit{not}(b) \wedge \varphi$ *and* $\varphi_{i+1} = (b \wedge wp(s, \varphi_i)) \vee (\mathit{not}(b) \wedge \varphi_i)$*.*

Assertions are special infinite first-order formulae in which only finitely many variables may occur. Hence, for reasons of consistency, the claim made in Definition 4.1 that for any while-program $p \in \mathcal{PRG}$ and assertion $\varphi \in \mathcal{ASS}$ the

formula $wp(p,\varphi) \in \mathcal{L}_{\omega_1\omega}$ is also an assertion remains to be shown. In the proof of the following Lemma 4.2 and also in the remainder of this article we assume (as generally done in the literature) that *a variable of a while-program (a so-called programming variable) does not occur bound in an assertion*. Due to this assumption we do not need a separation of the set X into programming variables and logical variables and further functions, usually called assignments, which then map the logical variables to semantic elements; but strictly speaking it has to be shown that the assumption remains true when applying the function wp.

Lemma 4.2 *Assume a while-program $p \in \mathcal{PRG}$ and an assertion $\varphi \in \mathcal{ASS}$. Then $var(wp(p,\varphi)) \subseteq var(p) \cup var(\varphi)$. As a consequence, $var(wp(p,\varphi))$ is finite, i.e., $wp(p,\varphi) \in \mathcal{ASS}$.*

Proof. We use structural induction on p.

The only interesting case of the induction base is that the program p is an assignment statement $x := t$. Here we have

$$var(wp(x := t, \varphi)) = var(def[t]) \cup var(\varphi[t/x]) \subseteq var(x := t) \cup var(\varphi)\,,$$

where the equation follows from the definition of wp and var and the inclusion uses that x does not occur bound in φ and no bound variable of φ occurs in the expression t, i.e. substitution works without introducing a fresh variable.

Also within the induction step we deal only with the interesting case of the program p being a while-loop `while` b `do` s `od`. We use a side induction and show

$$var(\varphi_i) \subseteq var(\texttt{while}\ b\ \texttt{do}\ s\ \texttt{od}) \cup var(\varphi) \qquad (*)$$

for all formulae φ_i, $i \in \mathbb{N}$, used in Definition 4.1 since in combination with the definition of the functions wp and var this implies

$$var(wp(\texttt{while}\ b\ \texttt{do}\ s\ \texttt{od}, \varphi)) = \bigcup_{i\in\mathbb{N}} var(\varphi_i) \subseteq var(\texttt{while}\ b\ \texttt{do}\ s\ \texttt{od}) \cup var(\varphi)\,.$$

The induction base $i = 0$ of the side induction follows from

$$var(\varphi_0) = var(not(b) \wedge \varphi) \subseteq var(\texttt{while}\ b\ \texttt{do}\ s\ \texttt{od}) \cup var(\varphi)$$

using the definitions of the functions wp and var and the induction step of the side induction is shown by

$$\begin{aligned}
var(\varphi_{i+1}) &= var(b) \cup var(wp(s,\varphi_i)) \cup var(\varphi_i) && \text{def. } wp, var\\
&\subseteq var(b) \cup var(s) \cup var(\varphi_i) && \text{ind. hyp. 4.2}\\
&\subseteq var(\texttt{while}\ b\ \texttt{do}\ s\ \texttt{od}) \cup var(\varphi) && \text{ind. hyp. } (*), \text{ def. } var
\end{aligned}$$

which concludes the proof. □

Since substitution in expressions is unique and due to Lemma 4.2, the function wp introduced in Definition 4.1 is well-defined. In the following we show that it in fact computes weakest preconditions as introduced in Definition 3.2.b, i.e., the wp-calculus given by Definition 4.1 and the provable so-called healthiness criteria (see below) is sound. We start the proof of soundness with the following lemma which constitutes the main part of the entire proof. Given $\sigma \in \mathsf{State}$, it describes the validity of an assertion of the form $wp(p, \varphi)$ according to σ in terms of the denotational semantics $[\![p]\!](\sigma)$ and the validity of φ according to this semantics.

Lemma 4.3 *For all while-programs $p \in \mathcal{PRG}$, assertions $\varphi \in \mathcal{ASS}$ and states $\sigma \in \mathsf{State}$ we have $\models wp(p, \varphi)[\sigma]$ iff $\bot \notin [\![p]\!](\sigma)$ and $\models\!\!\models \varphi[[\![p]\!](\sigma)]$.*

Proof. The proof is done by structural induction on p.

The induction base consists of three cases: Suppose the program p to be the empty statement `skip`. Then we have

	$\models wp(\mathtt{skip}, \varphi)[\sigma]$	
iff	$\models \varphi[\sigma]$	def. wp
iff	$\bot \notin \{\sigma\}$ and $\models\!\!\models \varphi[\{\sigma\}]$	$\sigma \in \mathsf{State}$, def. $\models\!\!\models$
iff	$\bot \notin [\![\mathtt{skip}]\!](\sigma)$ and $\models\!\!\models \varphi[[\![\mathtt{skip}]\!](\sigma)]$	def. semantics.

If p is the undefined statement `abort`, then we get $\not\models wp(\mathtt{abort}, \varphi)[\sigma]$ from the definition of the function wp and $\bot \in [\![\mathtt{abort}]\!](\sigma)$ from the denotational semantics. As a consequence, the equivalence to be shown is true since its both sides are false.

Finally within the induction base, assume the program p to be an assignment statement $x := t$ with x and t being of sort m. In this situation we obtain

	$\models wp(x := t, \varphi)[\sigma]$	
iff	$\models def[t][\sigma]$ and $\models \varphi[t/x][\sigma]$	def. wp, $\models$
iff	$\bot_m \neq [\![t]\!](\sigma)$ and $\models \varphi[\sigma[x \leftarrow [\![t]\!](\sigma)]]$	def. $\models$, subst. prop.
iff	$\{\bot\} \neq [\![x := t]\!](\sigma)$ and $\models\!\!\models \varphi[\{\sigma[x \leftarrow [\![t]\!](\sigma)]\}]$	def. semantics, $\models\!\!\models$
iff	$\bot \notin [\![x := t]\!](\sigma)$ and $\models\!\!\models \varphi[[\![x := t]\!](\sigma)]$	def. semantics.

The induction step consists of four cases. We start with the program p being a nondeterministic choice $s_1 \,[\!]\, s_2$. Then we obtain the desired equivalence

by

$$
\begin{array}{lll}
 & \models wp(s_1 \parallel s_2, \varphi)[\sigma] & \\
\text{iff} & \models wp(s_1, \varphi) \text{ and } \models wp(s_2, \varphi)[\sigma] & \text{def. } wp, \models \\
\text{iff} & \bot \notin [\![s_1]\!](\sigma) \text{ and } \models \varphi[[\![s_1]\!](\sigma)] \text{ and} & \\
 & \bot \notin [\![s_2]\!](\sigma) \text{ and } \models \varphi[[\![s_2]\!](\sigma)] & \text{ind. hyp. 4.3} \\
\text{iff} & \bot \notin [\![s_1]\!](\sigma) \cup [\![s_2]\!](\sigma) \text{ and } \models \varphi[[\![s_1]\!](\sigma) \cup [\![s_2]\!](\sigma)] & \text{def. } \models \\
\text{iff} & \bot \notin [\![s_1 \parallel s_2]\!](\sigma) \text{ and } \models \varphi[[\![s_1 \parallel s_2]\!](\sigma)] & \text{def. semantics.}
\end{array}
$$

Next we assume the program p to be a conditional $\texttt{if}\ b\ \texttt{then}\ s_1\ \texttt{else}\ s_2\ \texttt{fi}$. If $[\![b]\!](\sigma) = tt$, then we get $\models b[\sigma]$ and $\not\models not(b)[\sigma]$ which in turn implies

$$
\begin{array}{lll}
 & \models wp(\texttt{if}\ldots\texttt{fi}, \varphi)[\sigma] & \\
\text{iff} & \models wp(s_1, \varphi)[\sigma] & \text{def. } wp, \models \\
\text{iff} & \bot \notin [\![s_1]\!](\sigma) \text{ and } \models \varphi[[\![s_1]\!](\sigma)] & \text{ind. hyp. 4.3} \\
\text{iff} & \bot \notin [\![\texttt{if}\ldots\texttt{fi}]\!](\sigma) \text{ and } \models \varphi[[\![\texttt{if}\ldots\texttt{fi}]\!](\sigma)] & \text{def. semantics.}
\end{array}
$$

The case $[\![b]\!](\sigma) = ff$ is shown in the same way. Finally, if $[\![b]\!](\sigma) = \bot_{bool}$, then we have $\not\models b[\sigma]$ as well as $\not\models not(b)[\sigma]$. I.e., the left-hand side of the equivalence to be shown is false due to the definition of the function wp and the relation $\models$. From the denotational semantics we get $\bot \in [\![\texttt{if}\ldots\texttt{fi}]\!](\sigma)$, hence also the right-hand side of the equivalence to be shown is false.

The third case of the induction step is that the program p is a composed statement $s_1; s_2$. From the denotational semantics we obtain $\bot \in [\![s_2]\!](\bot)$. I.e., $\bot \in [\![s_1]\!](\sigma)$ implies the existence of a state $\rho \in [\![s_1]\!](\sigma)$ such that $\bot \in [\![s_2]\!](\rho)$.

Using the negation of this implication we get the claim by

$$
\begin{array}{lll}
 & \models wp(s_1; s_2, \varphi)[\sigma] & \\
\text{iff} & \models wp(s_1, wp(s_2, \varphi))[\sigma] & \text{def. } wp \\
\text{iff} & \bot \notin [\![s_1]\!](\sigma) \text{ and } \models\!\!\!\equiv wp(s_2, \varphi)[[\![s_1]\!](\sigma)] & \text{ind. hyp. 4.3} \\
\text{iff} & \bot \notin [\![s_1]\!](\sigma) \text{ and } \models wp(s_2, \varphi)[\rho] \text{ for all } \rho \in [\![s_1]\!](\sigma) & \text{def. } \models\!\!\!\equiv \\
\text{iff} & \bot \notin [\![s_1]\!](\sigma) \text{ and} & \\
 & \bot \notin [\![s_2]\!](\rho) \text{ and } \models\!\!\!\equiv \varphi[[\![s_2]\!](\rho)] \text{ for all } \rho \in [\![s_1]\!](\sigma) & \text{ind. hyp. 4.3} \\
\text{iff} & \bot \notin [\![s_2]\!](\rho) \text{ and } \models\!\!\!\equiv \varphi[[\![s_2]\!](\rho)] \text{ for all } \rho \in [\![s_1]\!](\sigma) & \text{see above} \\
\text{iff} & \bot \notin [\![s_2]\!](\rho) \text{ for all } \rho \in [\![s_1]\!](\sigma) \text{ and} & \\
 & \models\!\!\!\equiv \varphi[[\![s_2]\!](\rho)] \text{ for all } \rho \in [\![s_1]\!](\sigma) & \\
\text{iff} & \bot \notin \bigcup_{\rho \in [\![s_1]\!](\sigma)} [\![s_2]\!](\rho) \text{ and } \models\!\!\!\equiv \varphi[\bigcup_{\rho \in [\![s_1]\!](\sigma)} [\![s_2]\!](\rho)] & \text{def. } \models\!\!\!\equiv \\
\text{iff} & \bot \notin [\![s_1; s_2]\!](\sigma) \text{ and } \models\!\!\!\equiv \varphi[[\![s_1; s_2]\!](\sigma)] & \text{def. semantics.}
\end{array}
$$

Finally we suppose the program p to be a while-loop `while` b `do` s `od`. In this case, first, we use a side induction and show

$$\models \varphi_i[\rho] \quad \text{iff} \quad \bot \notin \tau^{i+1}[\Omega](\rho) \text{ and } \models\!\!\!\equiv \varphi[\tau^{i+1}[\Omega](\rho)] \qquad (*)$$

for all $\rho \in \mathsf{State}$ and all $i \in \mathbb{N}$, where the assertions φ_i, $i \in \mathbb{N}$, come from Definition 4.1 and the functional τ comes from Definition 2.2.

Due to the definition of the assertions φ_i, $i \in \mathbb{N}$, the two validity relations $\models$ and $\models\!\!\!\equiv$, and the functional τ, in case of the induction base $i = 0$ the equivalence $(*)$ to be shown reduces to the equivalence

$$\models not(b)[\rho] \text{ and } \models \varphi[\rho] \quad \text{iff} \quad \bot \notin \tau[\Omega](\rho) \text{ and } \models\!\!\!\equiv \varphi[\tau[\Omega](\rho)]\,.$$

This property holds. From $[\![b]\!](\rho) = tt$ or $[\![b]\!](\rho) = \bot_{bool}$ it follows that both sides are false and in the remaining case $[\![b]\!](\rho) = ff$ both sides are equivalent to $\models \varphi[\rho]$.

For a proof of the induction step $i \mapsto i+1$ of the side induction we consider again the three cases of the induction base. First, we assume that $[\![b]\!](\rho) = tt$.

Since then $\models b[\rho]$ and $\not\models not(b)[\rho]$ we obtain

$$
\begin{array}{lll}
 & \models \varphi_{i+1}[\rho] & \\
\text{iff} & \models wp(s,\varphi_i)[\rho] & \text{def. } \varphi_{i+1},\ \models \\
\text{iff} & \bot \notin [\![s]\!](\rho) \text{ and } \models\!\!\equiv \varphi_i[[\![s]\!](\rho)] & \text{ind. hyp. 4.3} \\
\text{iff} & \bot \notin [\![s]\!](\rho) \text{ and } \models \varphi_i[\gamma] \text{ for all } \gamma \in [\![s]\!](\rho) & \text{def. } \models\!\!\equiv \\
\text{iff} & \bot \notin [\![s]\!](\rho) \text{ and} & \\
 & \bot \notin \tau^{i+1}[\Omega](\gamma) \text{ and } \models\!\!\equiv \varphi[\tau^{i+1}[\Omega](\gamma)] \text{ for all } \gamma \in [\![s]\!](\rho) & \text{ind. hyp. } (*) \\
\text{iff} & \bot \notin \tau^{i+1}[\Omega](\gamma) \text{ and } \models\!\!\equiv \varphi[\tau^{i+1}[\Omega](\gamma)] \text{ for all } \gamma \in [\![s]\!](\rho) & \tau[h] \text{ strict} \\
\text{iff} & \bot \notin \tau^{i+1}[\Omega](\gamma) \text{ for all } \gamma \in [\![s]\!](\rho) \text{ and} & \\
 & \models\!\!\equiv \varphi[\tau^{i+1}[\Omega](\gamma)] \text{ for all } \gamma \in [\![s]\!](\rho) & \\
\text{iff} & \bot \notin \bigcup_{\gamma\in[\![s]\!](\rho)} \tau^{i+1}[\Omega](\gamma) \text{ and } \models\!\!\equiv \varphi[\bigcup_{\gamma\in[\![s]\!](\rho)} \tau^{i+1}[\Omega](\gamma)] & \text{def. } \models\!\!\equiv \\
\text{iff} & \bot \notin \tau^{i+2}[\Omega](\rho) \text{ and } \models\!\!\equiv \varphi[\tau^{i+2}[\Omega](\rho)] & \text{def. } \tau.
\end{array}
$$

In the second case $[\![b]\!](\rho) = \mathit{ff}$ we have $\models not(b)[\rho]$ and $\not\models b[\rho]$ and get the equivalence $(*)$ by the calculation

$$
\begin{array}{lll}
 & \models \varphi_{i+1}[\rho] & \\
\text{iff} & \models \varphi_i[\rho] & \text{def. } \varphi_{i+1},\ \models \\
\text{iff} & \bot \notin \tau^{i+1}[\Omega](\rho) \text{ and } \models\!\!\equiv \varphi[\tau^{i+1}[\Omega](\rho)] & \text{ind. hyp. } (*) \\
\text{iff} & \bot \notin \tau^{i+2}[\Omega](\rho) \text{ and } \models\!\!\equiv \varphi[\tau^{i+2}[\Omega](\rho)] & \tau^{i+2}[\Omega](\rho) = \{\rho\} = \tau^{i+1}[\Omega](\rho).
\end{array}
$$

In the remaining case $[\![b]\!](\rho) = \bot_{bool}$, from the definition of the function wp, the relations $\models$ and $\models\!\!\equiv$, and the functional τ we obtain that both sides of the equivalence $(*)$ are false. This ends the proof of $(*)$.

Returning to the main induction we now consider the limit of the iteration used in the fixed point theorem for continuous functions over cpos. From the definition of the Egli-Milner-order we obtain that

$$\bot \notin \bigsqcup_{i\in\mathbb{N}} \tau^i[\Omega](\sigma) \quad \text{iff} \quad \text{there exists } i_0 \in \mathbb{N} \text{ such that } \bot \notin \tau^{i_0}[\Omega](\sigma).$$

In this case we additionally have $\bigsqcup_{i\in\mathbb{N}} \tau^i[\Omega](\sigma) = \tau^{i_0}[\Omega](\sigma)$. With the help of these properties we are able to proceed by

$$
\begin{array}{lll}
 & \models wp(\texttt{while}\ldots\texttt{od},\varphi)[\sigma] & \\
\text{iff} & \models (\bigvee\{\varphi_i : i\in\mathbb{N}\})[\sigma] & \text{def. } wp \\
\text{iff} & \text{There exists } i\in\mathbb{N} \text{ such that } \models \varphi_i[\sigma] & \text{def. } \models \\
\text{iff} & \text{There exists } i\in\mathbb{N} \text{ such that} & \\
 & \bot \notin \tau^{i+1}[\Omega](\sigma) \text{ and } \models\!\!\models \varphi[\tau^{i+1}[\Omega](\sigma)] & \text{equiv. } (*) \\
\text{iff} & \bot \notin \bigsqcup_{i\in\mathbb{N}} \tau^{i+1}[\Omega](\sigma) \text{ and } \models\!\!\models \varphi[\bigsqcup_{i\in\mathbb{N}} \tau^{i+1}[\Omega](\sigma)] & \text{see above} \\
\text{iff} & \bot \notin \bigsqcup_{i\in\mathbb{N}} \tau^{i}[\Omega](\sigma) \text{ and } \models\!\!\models \varphi[\bigsqcup_{i\in\mathbb{N}} \tau^{i}[\Omega](\sigma)] & \\
\text{iff} & \bot \notin \mu_\tau(\sigma) \text{ and } \models\!\!\models \varphi[\mu_\tau(\sigma)] & \tau \text{ cont.} \\
\text{iff} & \bot \notin [\![\texttt{while}\ldots\texttt{od}]\!](\sigma) \text{ and } \models\!\!\models \varphi[[\![\texttt{while}\ldots\texttt{od}]\!](\sigma)] & \text{def. sem.}
\end{array}
$$

The transformation of the index in the fifth step of this derivation follows from the fact that $\tau^0[\Omega](\sigma) = \{\bot\}$ is the least element of the cpo $\mathbf{P}[\mathsf{State}^\bot]$. □

An immediate consequence of this lemma is: Given $\sigma \in \mathsf{State}$, then the specific assertion $wp(p, true)$ is valid according to this state iff $\bot$ is not contained in the denotational semantics $[\![p]\!](\sigma)$, i.e., iff every execution of the program p which starts with σ terminates.

Now we prove the main result of this section saying that the function wp in fact computes a weakest precondition according Definition 3.2.b. This proof is very easy since the difficult computations already have been carried out in the proof of the preceeding lemma.

Theorem 4.4 (Soundness) *The assertion $wp(p,\varphi) \in \mathcal{ASS}$ is a weakest precondition for total correctness according to the program $p \in \mathcal{PRG}$ and the postcondition $\varphi \in \mathcal{ASS}$.*

Proof. Let $\sigma \in \mathsf{State}$ be given. Using direction "⇒" of Lemma 4.3 we get

$$\models wp(p,\varphi)[\sigma] \quad \text{implies} \quad \bot \notin [\![p]\!](\sigma) \text{ and } \models\!\!\models \varphi[[\![p]\!](\sigma)]\,.$$

As an immediate consequence we have $[wp(p,\varphi)]\,p\,[\varphi]$ which is condition (i) of Definition 3.2.b. Now we assume a further assertion $\psi \in \mathcal{ASS}$ fulfilling

$[\psi]\, p\, [\varphi]$. Then we obtain $\models (\psi \to wp(p,\varphi))[\sigma]$ for all $\sigma \in \mathsf{State}$ from

$$\begin{array}{lll} & \models \psi[\sigma] & \\ \text{implies} & \bot \notin [\![p]\!](\sigma) \text{ and } \not\equiv \varphi[[\![p]\!](\sigma)] & \text{since } [\psi]\, p\, [\varphi] \\ \text{implies} & \models wp(p,\varphi)[\sigma] & \text{Lemma 4.3, direction ``}\Leftarrow\text{''.} \end{array}$$

Now, the unary form of $\models$ shows property (ii) of Definition 3.2.b. □

The soundness theorem expresses the fact that the set $\mathcal{L}_{\omega_1\omega}$ of infinite first-order formulae can be used to axiomatize weakest preconditions for total correctness. There are some other syntactical formalizations of the *wp*-calculus using, for example, weak second-order logic [23,6,7] or polymorphic higher-order logic [4]. But the set $\mathcal{L}$ of ordinary finite first-order formulae is too weak for formalizing weakest preconditions. A counter-example p over groups (due to R. Back [2]) is

$$\begin{array}{l} y := x; \\ \texttt{while } y \neq e \texttt{ do} \\ \quad y \;:=\; y \circ x \texttt{ od}, \end{array}$$

where the constant symbol e denotes the group's $(G, \cdot, 1)$ unit element 1 and the binary function symbol $\circ$ is interpreted as group multiplication $\cdot$. Given $\sigma \in \mathsf{State}$, it can easily be shown that $\models wp(p, \mathit{true})[\sigma]$ iff $\sigma(x) \neq \bot$ and there exists $i \in \mathbb{N} \setminus \{0\}$ such that $\sigma(x)^i$ equals the unit element, i.e., iff $\sigma(x)$ is a so-called torsion element. However it is known (see [13] for example) that this property cannot be described by a finite first-order formula.

In the last theorem of this section we show the strictness of the function *wp* and that it behaves well under conjunction. In the literature these properties are called healthiness criteria. Their proofs use only Lemma 4.3 in combination with the definitions of $\models$ and $\not\equiv$.

Theorem 4.5 (Healthiness Criteria) *The following properties hold for every while-program $p \in \mathcal{PRG}$ and each pair $\psi, \varphi \in \mathcal{ASS}$ of assertions:*

(a) $\models wp(p, \mathit{false}) \leftrightarrow \mathit{false}$ (b) $\models wp(p,\psi) \wedge wp(p,\varphi) \leftrightarrow wp(p, \psi \wedge \varphi)$

Proof. (a) We have for all $\sigma \in \mathsf{State}$ that

$$\begin{array}{lll} & \models wp(p, \mathit{false})[\sigma] & \\ \text{iff} & \bot \notin [\![p]\!](\sigma) \text{ and } \not\equiv \mathit{false}[[\![p]\!](\sigma)] & \text{Lemma 4.3} \\ \text{iff} & \bot \notin [\![p]\!](\sigma) \text{ and } \not\models \mathit{false}[\rho] \text{ for all } \rho \in [\![p]\!](\sigma) & \text{def. } \not\equiv \\ \text{iff} & \not\models \mathit{false}[\sigma] & \text{def. } \not\models, [\![\rho]\!](\sigma) \neq \emptyset\,, \end{array}$$

which implies $\models (wp(p, false) \leftrightarrow false)[\sigma]$ using the definition of $\models$. From this and the unary form of $\models$ the claim follows.

(b) Let $\sigma \in \mathsf{State}$ be given. Then we get

$$
\begin{array}{lll}
 & \models (wp(p,\psi) \wedge wp(p,\varphi))[\sigma] & \\
\text{iff} & \models wp(p,\psi)[\sigma] \text{ and } \models wp(p,\varphi)[\sigma] & \text{def. } \models \\
\text{iff} & \bot \notin [\![p]\!](\sigma) \text{ and } \models \psi[[\![p]\!](\sigma)] \text{ and } \models \varphi[[\![p]\!](\sigma)] & \text{Lemma 4.3} \\
\text{iff} & \bot \notin [\![p]\!](\sigma) \text{ and} & \\
 & \models \psi[\rho] \text{ for all } \rho \in [\![p]\!](\sigma) \text{ and } \models \varphi[\rho] \text{ for all } \rho \in [\![p]\!](\sigma) & \text{def. } \models \\
\text{iff} & \bot \notin [\![p]\!](\sigma) \text{ and } \models \psi[\rho] \text{ and } \models \varphi[\rho] \text{ for all } \rho \in [\![p]\!](\sigma) & \\
\text{iff} & \bot \notin [\![p]\!](\sigma) \text{ and } \models (\psi \wedge \varphi)[\rho] \text{ for all } \rho \in [\![p]\!](\sigma) & \text{def. } \models \\
\text{iff} & \bot \notin [\![p]\!](\sigma) \text{ and } \models (\psi \wedge \varphi)[[\![p]\!](\sigma)] & \text{def. } \models \\
\text{iff} & \models wp(p,\psi \wedge \varphi)[\sigma] & \text{Lemma 4.3}
\end{array}
$$

which implies $\models (wp(p,\psi) \wedge wp(p,\varphi) \leftrightarrow wp(p,\psi \wedge \varphi))[\sigma]$ using again of the definition of $\models$. The rest is as in (a). □

Besides strictness and conjunctivity in the literature one finds as further healthiness criteria disjunctivity

$$\models wp(p,\psi) \vee wp(p,\varphi) \rightarrow wp(p,\psi \vee \varphi)$$

and monotonicity

$$\models \psi \rightarrow \varphi \quad \text{implies} \quad \models wp(p,\psi) \rightarrow wp(p,\varphi)\,.$$

These, however, easily follow from Lemma 4.5 and logic. Sometimes also continuity of wp is called a healthiness criterion. In our purely syntactical approach continuity means that for all countable infinite sets $\{\varphi_i : i \in \mathbb{N}\}$ of assertions

$$
\begin{array}{ll}
 & \models \varphi_i \rightarrow \varphi_{i+1} \text{ for all } i \in \mathbb{N} \\
\text{implies} & \models wp(p, \bigvee\{\varphi_i : i \in \mathbb{N}\}) \leftrightarrow \bigvee\{wp(p,\varphi_i) : i \in \mathbb{N}\}\,.
\end{array}
$$

Continuity expresses the fact that nondeterminism is bounded, i.e., operationally all computation trees are only finitely branching. Following the pattern of the last proof, also this property easily can be shown just as further properties of the function wp like its distributivity over universal quantification (mentioned in [6,7]), subdistributivity over existential quantification,

the equivalence of $[\psi]\ p\ [\varphi]$ and $\models \psi \rightarrow wp(p, \varphi)$, and the well-known loop invariant theorem [14] for estimating weakest preconditions of while-loops.

5 Conclusion

Our main concern was to consider an established and well proved approach to weakest preconditions (which also is very close to the usual semantical approach presented in the most well-known textbooks) and to close wrt. it a gap in the theory. Based on the work of R. Back, we have presented a purely syntactical definition of E.W. Dijkstra's weakest preconditions predicate transformer as a function *wp* from programs and formulae to formulae and have shown that this definition is sound wrt. a definition of weakest preconditions given in terms of denotational semantics and the validity of formulae according to states.

The relationship between axiomatic and denotational semantics established by this result allows to use both notions in a mutual way. In particular, given a specific problem one can use the more qualified semantics to solve it. First examples for such a proceeding have been the proofs of the two healthiness criteria. Without the connection to denotational semantics these require costly inductions on the structure of the given program. As a further example we consider a proof for a while-loop to terminate when a certain condition holds, a task which frequently occurs in program derivation resp. verification. Formally this means that we have to show $\models (\psi \rightarrow wp(\texttt{while}\ \mathrm{b}\ \texttt{do}\ \mathrm{s}\ \texttt{od}, \mathit{true}))[\sigma]$ for all $\sigma \in \mathsf{State}$. In our experience the easiest approach to prove this property *formally* is, first, to define a Noetherian order on the set $\{\sigma \in \mathsf{State} : \models \psi[\sigma]\}$ and, then, to prove for all states σ from this set the inequation $\mu_\tau(\sigma) \neq \bot$ (where τ comes from Definition 2.2) by Noetherian induction using denotational semantics. Of course, this approach is similar to the well-known method (presented in [14] for example) using a so-called bound function. However, in contrast with [14], it takes also into consideration so-called finite errors like undefined loop conditions.

In this article we have been concerned only with erratic nondeterminism since this seems to be the most import kind of nondeterminism, in particular wrt. implementability. Following the lines of the presented proof it is obvious that the *wp*-calculus is also sound if we define denotational semantics in such a way that nondeterminism becomes demonic, which means that possible nontermination is equivalent to guaranteed nontermination. The third kind of nondeterminism discussed in the literature, called angelic, only considers partial correctness since here possible termination is equivalent to guaranteed termination; a comparison with weakest preconditions for total correctness seems not to be reasonable. Some arguments for an erratic approach to program semantics being better than a demonic or angelic approach can be found in [12] for example.

A serious assumption, however, is that in our programming language non-

determinism is bounded. For two reasons it seems to be difficult to generalize our proof of soundness to a programming language allowing also unbounded nondeterminism. First, in this case continuity of the loop functional τ does not hold in general such that the fixed point theorem for continuous functions cannot be applied. And, second, the infinitary first-order logic $\mathcal{L}_{\omega_1\omega}$ is too weak to express weakest preconditions for unbounded nondeterministic programs as shown by R. Back [2] using that well-foundedness cannot be characterized in $\mathcal{L}_{\omega_1\omega}$ [18]. There are some approaches to weakest preconditions which deal with unbounded nondeterminism, but all of them are of semantical nature or use the (sometimes dangerous) common practice mentioned in the introduction. See [1,3,8] for example. Using a first-order logic enriched by a least fixed point operator, perhaps a purely syntactical formalization can be obtained. (For deterministic programs this was sketched to the author by K. Apt.) The difficulties here are that one has to consider syntactical monotonicity (i.e., the number of negations over a variable) and implication leads on formulae only to a pre-order such that least elements are not unique. Furthermore, the question remains whether for a practical use such a fixed point formalization is superior to that of this article, which bases on a for years well-proved approach.

Acknowledgement

This article benefited from valuable discussions with Eike Best, Lex Bijlsma, Martin Fränzle, and Markus Müller-Olm.

References

[1] Apt K.R., Plotkin G.D.: Countable nondeterminism and random assignment. Jounal ACM 13, 724-767 (1986)

[2] Back, R.-J.: Proving total correctness of nondeterministic programs in infinitary logic. Acta Informatica 15, 233-249 (1981)

[3] Back, R.-J., von Wright J.: Duality in specification languages: A lattice-theoretic approach. Acta Informatica 27, 583-625 (1990)

[4] Back, R.-J., von Wright J.: Predicate transformers and higher order logic. In: de Bakker J.W., de Roever W.-P., Rozenberg G. (eds.): Proc. REX Workshop "Semantics: Foundations and Applications", Beekbergen (Niederlande) 1992, LNCS 666, Springer, 1-20 (1993)

[5] Backhouse R.C.: Program construction and verification. Prentice-Hall (1986)

[6] Berghammer R., Elbl B., Schmerl U.: Proving correctness of programs in weak second-order logic. In: de Bakker J.W., de Roever W.-P., Rozenberg G. (eds.): Proc. REX Workshop "Semantics: Foundations and Applications", Beekbergen (Niederlande) 1992, LNCS 666, Springer, 51-72 (1993)

[7] Berghammer R., Elbl B., Schmerl U.: Formalizing Dijkstra's predicate transformer *wp* in weak second-order logic. Theoretical Computer Science 146, 185-197 (1995)

[8] Best E.: Semantik – Theorie sequentieller und paralleler Programmierung. Vieweg (1995)

[9] Dijkstra E.W.: Guarded commands, nondeterminacy and formal derivation of programs. Comm. ACM 18 (8), 453-457 (1975)

[10] Dijkstra E.W.: A discipline of programming. Prentice Hall (1976)

[11] Dijkstra E.W., Scholten C.S.: Predicate calculus and program semantics. Springer (1990)

[12] Doornbos H.: A relational model of programs without the restriction to Egli-Milner monotone constructs. In: Olderog E.-R. (ed.): Programming Concepts, Methods and Calculi (ProCoMet '94), IFIP Transactions A-56, 363-382, North-Holland (1994)

[13] Ebbinghaus H.-D., Flum J., Thomas W.: Mathematical logic. Springer (1994)

[14] Gries D.: The science of programming. Springer (1981)

[15] Harel D.: On folk theorems. Comm. ACM 23 (7), 379-389 (1980)

[16] Hoare C.A.R.: An axiomatic basis for computer programming. Comm. ACM 12 (10), 576-580, 583 (1969)

[17] Karp C.R.: Languages with expressions of infinite length. North-Holland (1964)

[18] Keisler H.J.: Model theory of infinitary logic. North-Holland (1971)

[19] Morgan C.: Programming from specifications. Prentice-Hall (1990)

[20] Morgan C., Vickers T. (eds.): On the refinement calculus. Springer (1992)

[21] Nielson H.R., Nielson F.: Semantics with applications. A formal introduction. Wiley (1992)

[22] Plotkin G.D.: A powerdomain construction. SIAM J. Comput 5, 452-486 (1976)

[23] Tucker J.V, Zucker J.L.: Program correctness over abstract data types with error-state semantics. CWI Monographs 6, North-Holland (1988)

[24] Winkler J.F.H: *wp* is basically a state set transformer. Institut für Informatik, Universität Jena, `http://www1.informatik.uni-jena.de/themen/pap-talk/pap-talk.htm` (1996)

[25] Winskel G.: The formal semantics of programming languages. An introduction. Series "Foundations of Computing", The MIT Press (1993)

[26] Wirsing M.: Algebraic Specification. In: van Leeuwen J. (ed.): Handbook of Theoretical Computer Science, Vol. B, Elsevier Science Publishers, 675-788 (1990)